Parallel Histogram Equalization Using CUDA

Erik Pahor, Jaka Škerjanc

I. Introduction

This project implements parallel histogram equalization for color images using CUDA. Histogram equalization is a contrast enhancement technique that redistributes image luminance to span the entire intensity range. For color images, we operate on the Y (luminance) channel after converting RGB to YUV, to avoid altering color tones.

The process includes: converting RGB to YUV, computing the luminance histogram, calculating the cumulative distribution function (CDF), equalizing pixel intensities, and converting the image back to RGB.

We implemented a sequential CPU version, a parallel CUDA version with atomic operations and an optimized CUDA version using shared memory and partial histograms to reduce contention.

All stages were offloaded to the GPU with minimal host-device memory transfers to maximize performance.

II. Implementation Details

A. RGB to YUV Conversion

This transformation is applied in parallel: one thread per pixel. Each thread computes Y, U, and V using matrix multiplication. Only the Y component is processed for equalization.

B. Histogram Computation

The baseline CUDA implementation uses a global histogram updated with `atomicAdd`. However, atomic contention becomes a bottleneck for high-resolution images.

To optimize this, we allocate shared memory per thread block to store partial histograms. Each thread updates the local shared histogram, reducing global memory atomics. After synchronization, block histograms are atomically merged into the global histogram.

C. CDF and Lookup Table

We first implemented a simple CDF computation using a single GPU thread. For the bonus version, we applied Blelloch's work-efficient scan algorithm to parallelize this stage, leveraging shared memory and thread synchronization.

The final lookup table (LUT) maps old luminance values to new ones using:

$$l' = \frac{CDF(l) - CDF_{min}}{(N \times M - CDF_{min})} \times (L - 1)$$

This LUT is stored in global memory and reused during intensity remapping.

D. Applying the LUT

Each pixel's Y value is replaced using the LUT. Threads operate independently, making this step highly parallelizable. The final YUV image is converted back to RGB in parallel.

III. Experimental Results

We tested the three implementations on images of various resolutions. Each result is averaged over 7 runs. Table I shows the average execution times.

Image Size	Sequential	Parallel	Optimized
720x480	14.60 ms	0.74 ms	0.47 ms
1024x768	29.25 ms	1.00 ms	0.75 ms
1920x1200	80.46 ms	3.14 ms	1.61 ms
3840x2160	274.34 ms	7.75 ms	4.04 ms
7680x4320	989.65 ms	31.53 ms	15.23 ms

Table I
Average execution times for each implementation.

The resulting speed-ups are presented in Table II.

Image Size	Parallel Speed-Up	Optimized Speed-Up
720x480	19.7×	31.0×
1024x768	29.3×	39.3×
1920x1200	25.6×	49.9×
3840x2160	35.4×	67.9×
7680x4320	31.4×	65.0×

Table II
Speed-up factors relative to the sequential version.

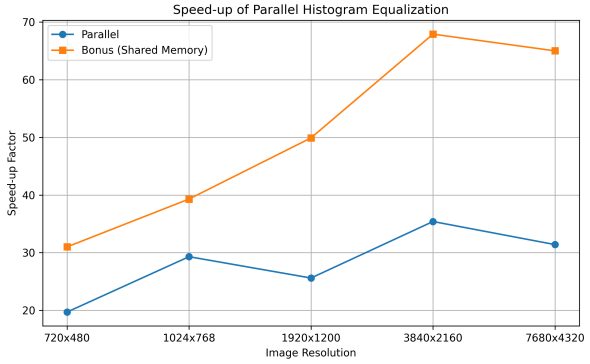

Figure 1. Speed-up of parallel and optimized CUDA implementations.

IV. Discussion

Significant speed-ups were observed, especially at higher resolutions where GPU parallelism is better utilized. The baseline CUDA version suffers from atomic contention in histogram computation. Introducing shared memory in the optimized version drastically reduced contention and improved memory coalescing.

The use of partial histograms per thread block greatly reduced global memory atomic operations. For the CDF stage, the parallel prefix sum (scan) reduced runtime compared to the naive implementation, though benefits here were modest due to the small size (256 bins).

Overall, optimizations targeting memory hierarchy and minimizing contention had the greatest impact on performance.

V. Conclusion

We demonstrated a full GPU-accelerated histogram equalization algorithm for color images using CUDA. Optimizations such as shared memory usage, partial histograms, and work-efficient scan significantly improved performance. For large images, our optimized version achieved speed-ups exceeding $65\times$ over the sequential version.

Future improvements could include overlapping computation and data transfer with CUDA streams or exploring warp-level primitives for even finer control.